Selected Topics on Java Programming

Unit Testing with Junit 4. Java 8 Stream API

Where to Find The Code and Materials?

https://github.com/iproduct/course-java-web-development

Unit Testing with Junit 4

Съдържание

Тестване на софтуера

Софтуерното тестване е процес на изследване на софтуера, с цел получаване на информация за качеството на продукта или услугата, която се изпитва. Софтуерното тестване може да осигури обективен, независим поглед, който да даде възможност на клиента да разбере рисковете при реализацията на софтуера. Техниките за тестване включват (но не са ограничени до) изпълнение на програмата с намерение да се открият софтуерни бъгове (грешки или други дефекти). Процесът на софтуерно тестване е неразделна част от софтуерното инженерство и осигуряване на качеството на софтуера.

[Wikipedia]

Източник : Wikipedia - http://en.wikipedia.org/wiki/Software_testing

Видове тестване

- Статично и динамично тестване
- White-Box testing – тества вътрешната структура и работа на софтуера (API testing, Code coverage, Fault injection – Stress testing, Mutation testing)
- Black-box testing – тества функционалността без да се интересува от вътрешната реализация

- Grey-box testing – използва познания за структурите от данни и алгоритмите при разработката на тестове
- Визуално тестване – записват се всички действия на тестващия с цел лесно да се възпроизведе проблема

Нива на тестване

- Unit testing – компонентно тестване, при което се тества функционалността на специфична секция от кода (обикновено метод – като минимум конструкторите)
- Integration testing – проверява дали интерфейсите между компонентите са реализирани според спецификацията им
- System testing – тества се напълно интегрираната система за да се определи дали реализацията съответства на изискванията
- Acceptance testing – тестване на системата от крайните й потребители

Специфични цели при тестване

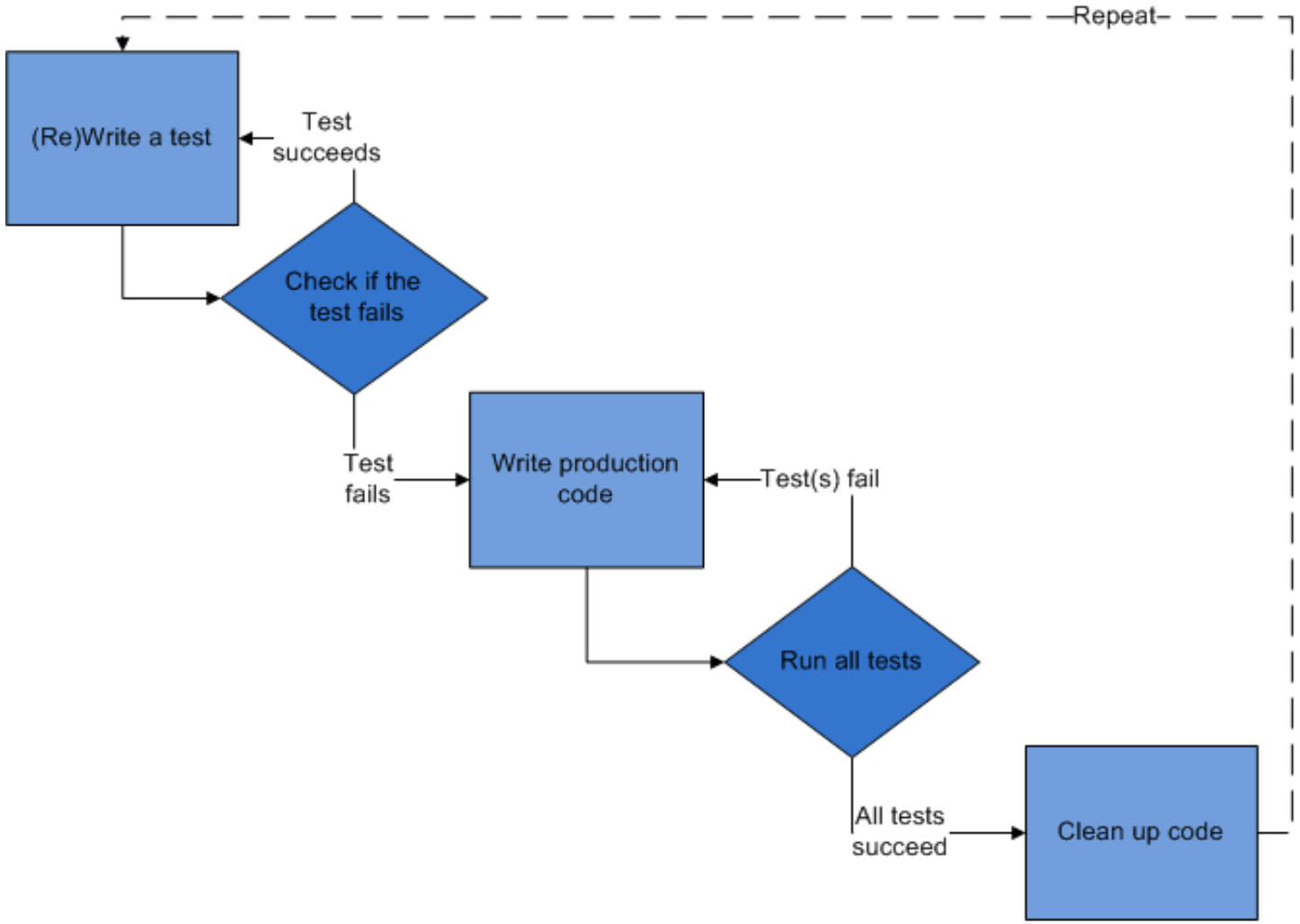
- Installation testing
- Compatibility testing
- Smoke and sanity testing
- Regression testing
- Acceptance testing
- Alpha testing
- Beta testing
- Functional vs non-functional testing
- Destructive testing
- Software performance testing
- Usability testing
- Accessibility
- Security testing
- Internationalization and localization
- Development testing

Източник : Wikipedia - http://en.wikipedia.org/wiki/Software_testing

Test Driven Development (TDD) с JUnit 4

- **Test-Driven Development (TDD)** е техника, при която разработката на софтуер се насочва чрез писане на тестове.
- Първоначално е развита от Kent Beck (в края на 90-те).
- Основната идея е да се повтарят последователно следните пет стъпки:
  1. Пишем автоматичен тест (Unit test) за следващата **малка** част нова функционалност като си представяме че кодът вече съществува;
  2. Пишем празни методи (Stubs), така че кодът да се компилира;
  3. Пускаме теста – той трябва да пропадне, иначе тестът не е добър;
  4. Пишем **минималното** количество функционален код така, че тестът да успее – ако тестът не минава успешно, значи кодът не е добър;
  5. Променяме (Refactor) както стария, така и новия код, за да го структурираме по-добре.

Последователни етапи при TDD



Източник: Wikipedia (http://en.wikipedia.org/wiki/File:Test-driven_development.PNG), Автор: Excirial, Лиценз: Creative Commons Attribution, ShareAlike 3.0 License / GNU Free Documentation License, Version 1.2

Agile Testing - TDD

Ето един добър начин за разработка на нова функционалност:

- Вижте какво имате да правите.
- Напишете UnitTest за желаната функционалност, като изберете най-малкия възможен инкремент, който ви хрумва.
- Стартирайте UnitTest-а. Ако е успешен сте готови; отидете на стъпка 1 или ако сте напълно готови си идете у дома.
- Решете текущия проблем: може би все още не сте написали новия метод. Може би методът не работи точно както трябва. Направете необходимите поправки. Отидете на стъпка 3.

Източник : http://c2.com/cgi/wiki?CodeUnitTestFirst

JUnit 4

- **UnitTest** – тества конкретен клас и за да може да го вижда се намира в същия пакет (ако искаме достъп до private членовете на класа можем да използваме рефлексия или да реализираме теста като вътрешен клас). В JUnit 3 беше необходимо той да наследява junit.framework.TestCase и всеки метод да се именува testXXX. В **JUnit 4** всички тези изисквания отпадат, а вместо това всеки тестов метод се анотира с **@Test**.
- **Test Suite** – дава възможност за обединяване на тестовете в комплекти от тестове:

```
@RunWith(Suite.class)

@SuiteClasses({ PriceComparatorTest.class, TransactionTest.class })

public class AllTests { }
```

Основни анотации в JUnit 4

@Test – идентифицира метода като тестов в JUnit 4

@Test (expected = ExtendingException.class)

@Test (timeout=200) – максимално време в милисекунди

@Before – изпълнява метода преди всеки тест

@After – изпълнява метода след всеки тест

@BeforeClass – изпълнява метода еднократно преди всички тестове – самият метод трябва да е статичен

@AfterClass – изпълнява метода еднократно след приключване на всички тестове – статичен метод

@Ignore – игнорира анотирания тестов метод

@RunWith – анотира Runner класа който изпълнява тестовете

@SuiteClasses – анотира класовете в тестови комплект (test suit) използва се в комбинация с **@RunWith(Suite.class)**

Шаблон за тестов случай (Test Case) с JUnit 4

```
import org.junit.Ignore;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.JUnit4;

/** Tests for {@link Foo}. */
@RunWith(JUnit4.class)
public class FooTest {
    @Test
    public void thisAlwaysPasses() {}

    @Test
    @Ignore
    public void thisIsIgnored() {}
}
```

Источник : https://github.com/junit-team/junit/wiki/Getting-started

Пример за JUnit 4 тестови случаи (1)

```
public class GcdTest {
   @Test
   public void testIsPositiveInteger() {
      String[] testData = {"346", "-23", "29a34", "17.5"};
      boolean[] resultData = {true, false, false, false };
      for(int i = 0; i < testData.length; i++){
         assertEquals(resultData[i],
            Gcd.isPositiveInteger(testData[i]));
      }
   } ...
```

Пример за JUnit 4 тестови случаи (2)

```
@Test
public void testGreatestCommonDenominator() {
    int [][] testData= {{48, 72, 24}, {17, 351, 1}, {81, 63, 9}};
    for(int[] data: testData){
        int result = Gcd.greatestCommonDenominator(data[0], data[1]);
        assertEquals(data[2], result);
        result = Gcd.greatestCommonDenominator(data[1],data[0]);
        assertEquals(data[2], result);
    }
}
```

Условия за валидност (Assertions)

```
import static  org.junit.Assert.*;
assertArrayEquals("values not same", expected, actual)
assertFalse("failure - should be false", expected)
assertTrue("failure - should be true", expected)
assertNotNull("should not be null", myObject)
assertNotSame("should not be same Object", myObject, other)
assertNull("should be null", null)
assertSame("should be same", aNumber, aNumber)
assertThat("good", not(allOf(equalTo("bad"), equalTo("good"))))  ...
```

Основни анотации в JUnit 4 и ред на изпълнение

1.@BeforeClass setupClass()

2.@Before setup()

3.@Test test1()

4.@After cleanup()

5.@Before setup()

6.@Test test2()

7.@After cleanup()

8.@AfterClass cleanupClass()

Пример - JUnit 4 (1)

```
public class TransactionTest {

private static InputStream in;

@BeforeClass

public static void setUpBeforeClass() throws Exception {

  String data="Goole Inc.\nJohn Smith\nGOGL\n42.78\n120\n";

  in = new ByteArrayInputStream(data.getBytes()));

}

@AfterClass

public static void tearDownAfterClass() throws Exception {

  in.close();

}
```

Пример - JUnit 4 (2)

```
@Test
public void testTransactionFullConstructor() {
  Transaction t = new Transaction("Google Inc.", "John Smith", "GOGL", 27,
  19.439834456455544);
  assertNotNull(t);
  assertTrue("Transaction ID not correct", t.getId() > 0);
  assertTrue("'timestamp' not correct",
            t.getTimestamp().getTime() <= new Date().getTime());
  assertEquals("Google Inc.", t.getSeller());
  assertEquals("John Smith", t.getBuyer());
  assertEquals("GOGL", t.getSymbol());
  assertEquals(27, t.getQuantity());
  assertEquals(19.4398, t.getPrice(),1E-4);
}
```

Пример - JUnit 4 (3)

```
  @Test
  public void testInput() {
    Transaction t = new Transaction();
    assertNotNull(t);
    t.input(in);
    assertTrue("Transaction ID not correct", t.getId()>0);
    assertEquals("Google Inc.", t.getSeller());
    assertEquals("John Smith", t.getBuyer());
    assertEquals("GOGL", t.getSymbol());
    assertEquals(120, t.getQuantity());
    assertEquals(42.78, t.getPrice(),1E-4);
  }
  ...
}
```

Пример за тестов комплект- JUnit 4

```
import org.junit.runner.RunWith;

import org.junit.runners.Suite;

import org.junit.runners.Suite.SuiteClasses;


@RunWith(Suite.class)

@SuiteClasses({ PriceComparatorTest.class, TransactionTest.class })

public class AllTests {

}
```

Параметризирани тестове с JUnit 4 (1)

```
@RunWith(value = Parameterized.class)
public class GcdTestWithParameters {
    private int numberA;
    private int numberB;
    private int expected;
    //pass parameters using test constructor
    public GcdTestWithParameters(
                        int numberA, int numberB, int expected) {
        this.numberA = numberA;
        this.numberB = numberB;
        this.expected = expected;
    }
```

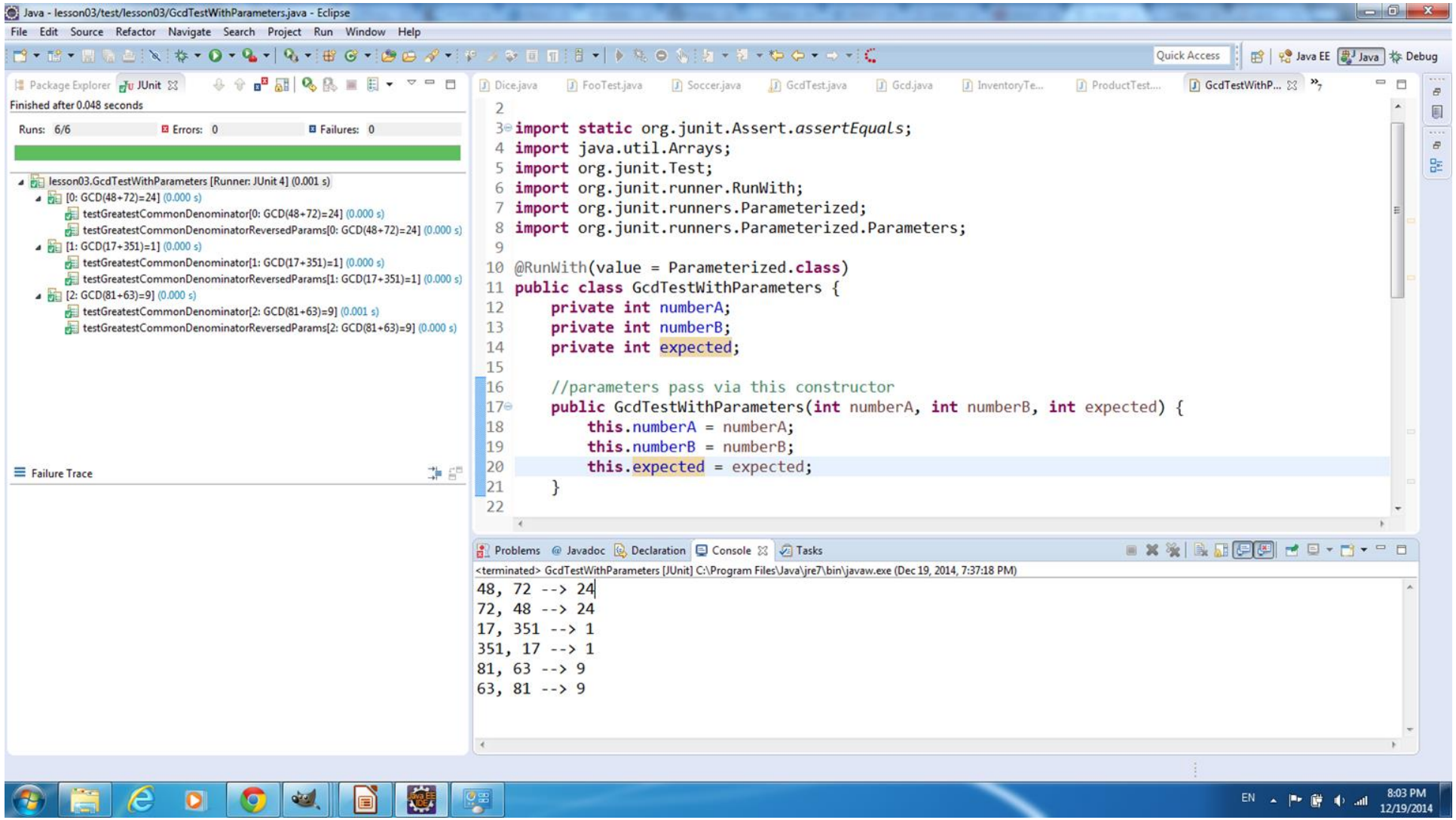
Параметризирани тестове с JUnit 4 (2)

```
//Declared test parameters
@Parameters(name = "{index}: GCD({0}+{1})={2}")
public static Iterable<Object[]> data1() {
    return Arrays.asList(new Object[][] {
        {48, 72, 24},
        {17, 351, 1},
        {81, 63, 9}
    });
}
```

Параметризирани тестове с JUnit 4 (3)

```
    @Test
    public void testGreatestCommonDenominator() {
        int result = Gcd.greatestCommonDenominator(numberA, numberB);
        assertEquals(expected, result);
    }
    @Test
    public void testGreatestCommonDenominatorReversedParams() {
        int result = Gcd.greatestCommonDenominator(numberB, numberA);
        assertEquals(expected, result);
    }
}
```

Параметризирани тестове с JUnit 4 (4)



Задача: TDD с Junit 4, параметрични тестове, Exception тестове

Решение в GitHub: https://github.com/iproduct/course-java-web-development/tree/master/unit-tests-lab2

Задача: (Шифриране и дешифриране на текст):

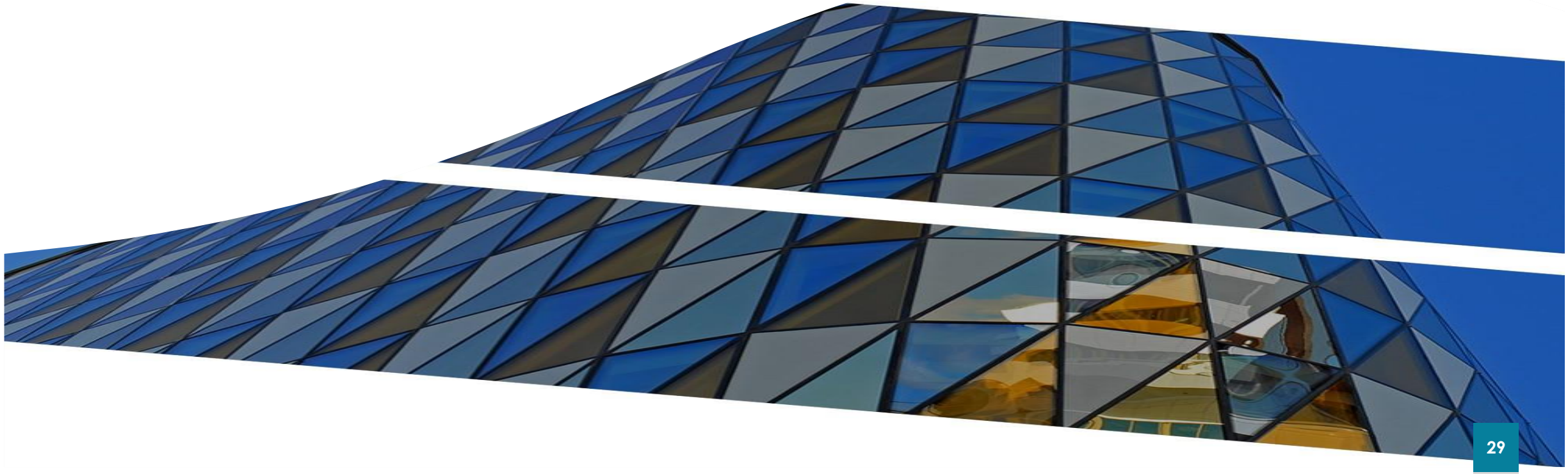
- Следвайки принципите на TDD, напишете програма, която по зададен низ **s** и цяло число (ключ) **k** кодира низа като събира кода на всяка буква от **s** с числото ключ **k** и отмества всяка от буквите **k** позиции надясно по модул дължината на низа (т.е. ако буквата излезе извън низа – започваме да броим оставащите позиции за отместване отново отляво от началото на низа). Реализирайте и метод за дешифриране на текста шифриран по гореописания алгоритъм.
- Реализирайте unit тестове за обработка на InvalidDataException изключение, генерирано при получаване на символ извън зададения интервал [0,1024]. Проверете test coverage – цел 100%.

Литература и интернет ресурси

- Software testing в Wikipedia - http://en.wikipedia.org/wiki/Software_testing
- Test-driven development в Wikipedia - http://en.wikipedia.org/wiki/Test-driven_development
- Test Driven Development wiki - http://c2.com/cgi/wiki?TestDrivenDevelopment
- Junit 4 - https://github.com/junit-team/junit/wiki
- JUnit 4 Tutorial - http://www.vogella.com/articles/JUnit/article.html

Java 8 Stream API

Practical Exercises – Functional Programming Koans



Agenda for This Session

- Fundamentals
- Functional interfaces
- Method references
- Constructor references

Новости в Java™ 8

- Ламбда изрази и поточно програмиране – пакети **java.util.function** и **java.util.stream**)
- Референции към методи
- Методи по подразбиране и статични методи в интерфейси – множествено наследяване на поведение в Java 8
- Функционално програмиране в Java 8 с използване на монади (напр. Optional, Stream) – предимства, начин на реализация, основни езикови идиоми, примери

Функционални интерфейси в Java™ 8

- Функционален интерфейс = интерфейс с един абстрактен метод
  SAM (Single Abstract Method) – @FunctionalInterface
- Примери за функционални интерфейси в Java 8:

```
public interface Comparator<T> {
    int compare(T o1, T o2);
}
public interface ActionListener extends EventListener {
    public void actionPerformed(ActionEvent e);
}
public interface Runnable {
    public void run();
}
public interface Callable<V> {
    V call() throws Exception;
}
```

Ламбда изрази – пакет java.util.function

Примери:

```
(int x, int y) -> x + y

() -> 42

(a, b) -> a * a + b * b;

(String s) -> { System.out.println(s); }

book -> book.getAuthor().fullName()

voter -> voter.getAge() >= legalAgeOfVoting

(person1, person2) -> person1.getAge() - person2.getAge()

(song1, song2) -> song1.getArtist().compareTo(song2.getArtist())
```

Правила за форматирне на ламбда изрази

- **Ламбда изразите (функциите)** могат да имат произволен брой **параметри**, които се ограждат в скоби, разделят се със запетаи и могат да имат или не деклариран тип (ако нямат - типът им се извежда от **контекста на използване = target typing**). Ако са само с един параметър, то скобите не са задължителни.
- **Тялото на ламбда изразите** се състои от произволен езикови конструкции (statements), разделени с **;** и заградени във фигурни скоби. Ако имаме само една езикова конструкция – израз то използването на фигурни скоби не е необходимо – в този случай стойността на израза автоматично се връща като стойност на функцията.

Пакет java.util.function

- **Predicate<T>** – предикат = булев израз представящ свойство на обекта подавано като аргумент
- **Function<A,R>**: функция която приема като аргумент **A** и го трансформира в резултат **R**
- **Supplier<T>** – с помощта на **get()** метод всеки път връща инстанция (обект) – фабрика за обекти
- **Consumer<T>** – приема аргумент (метод **accept()**) и изпълнява действие върху него
- **UnaryOperator<T>** – оператор с един аргумент **T -> T**
- **BinaryOperator<T>** – бинарен оператор **(T, T) -> T**

Поточно програмиране (1)

Примери:

```
books.stream().map(book ->
    book.getTitle()).collect(Collectors.toList());
books.stream()
       .filter(w -> w.getDomain() == PROGRAMMING)
       .mapToDouble(w -> w.getPrice()) .sum();
document.getPages().stream()
     .map(doc -> Documents.characterCount(doc))
     .collect(Collectors.toList());
document.getPages().stream()
     .map(p -> pagePrinter.printPage(p))
     .forEach(s -> output.append(s));
```

Поточно програмиране (2)

Примери:

```
document.getPages().stream()
        .map(page -> page.getContent())
        .map(content -> translator.translate(content))
        .map(translated -> new Page(translated))
        .collect(Collectors.collectingAndThen(
                Collectors.toList(),
                pages -> new
        Document(translator.translate(document.getTitle()), pages)));
```

Референции към методи

- Статични методи на клас – Class::staticMethod
- Методи на конкретни обектни инстанции – object::instanceMethod
- Методи на инстанции реферирани чрез класа – Class::instanceMethod
- Конструктори на обекти от даден клас – Class::new

```
Comparator<Person> namecomp = Comparator.comparing(Person::getName);

Arrays.stream(pageNumbers).map(doc::getPageContent).forEach(Printers::print);

pages.stream().map(Page::getContent).forEach(Printers::print);
```

Статични и Default методи в интерфейси

- Методите с реализация по подразбиране в интерфейс са известни още като virtual extension methods или defender methods, защото дават възможност интерфейсите да бъдат разширявани, без това да води до невъзможност за компилация на вече съществуващи реализации на тези интерфейси (което би се получило ако старите реализации не имплементират новите абстрактни методи).
- Статичните методи дават възможност за добавяне на помощни (utility) методи – например factory методи директно в интерфейсите които ги ползват, вместо в отделни помощни класове (напр. Arrays, Collections).

Пример за default и static методи в интерфейс

- @FunctionalInterface

```
interface Event {
    Date getDate();
    default String getDateFormatted() {
            return String.format("%1$td.%1$tm.%1$tY", getDate());
    }
    public static <T, U extends Comparable<? super U>>
    Comparator<T> comparing(Function<T, U> getKey) {
        return (c1, c2) -> getKey.apply(c1).compareTo(getKey.apply(c2));
    }
}
Event current = () -> new Date();
System.out.println(current.getDateFormatted());
```

Функционално програмиране и монади

- Понятие за **монада** във функционалното програмиране (теория на категориите) – **Монадата** е множество от три елемента:

- Параметризиран тип **M<T>**
- "**unit**" функция: **T -> M<T>**
- "**bind**" операция: **bind(M<T>, f:T -> M<U>) -> M<U>**

- В Java 8 пример за монада е класът **java.util.Optional<T>**

Параметризиран тип: **Optional<T>**

- "**unit**" функции: **Optional<T> of(T value) , Optional<T> ofNullable(T value)**
- "**bind**" операция: **Optional<U> flatMap(Function<? super T,Optional<U>> mapper)**

Задача: Java 8 Stream API Koans

Достъпна в GitHub: https://github.com/iproduct/course-java-web-development/tree/master/lambda-tutorial-master

1. Прочетете внимателно JavaDoc на unit тестовете описващ задачите за решаване: Exercise_1_Test.java, Exercise_2_Test.java, Exercise_3_Test.java, Exercise_4_Test.java and Exercise_5_Test.java
2. Попълнете кода на мястото на коментарите от типа:
   // [your code here]
3. Изпълнете unit тестовете за да проверите дали правилно сте решили задачата. Ако да – продължете със следващата задача. Ако не сте – отидете на стъпка 1.

Литература и интернет ресурси

- Oracle tutorial – lambda expressions - http://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html
- Java SE 8: Lambda Quick Start - http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/Lambda-QuickStart/index.html
- OpenJDK Lambda Tutorial - https://github.com/AdoptOpenJDK/lambda-tutorial

Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

http://robolearn.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD